Degenerate String Comparison and Applications

Mai Alzamel
Department of Informatics, King's College London, UK and Department of Computer Science, King Saud University, KSA
mai.alzamel@kcl.ac.uk

Lorraine A. K. Ayad
Department of Informatics, King's College London, UK
lorraine.ayad@kcl.ac.uk

Giulia Bernardini[1]
Department of Informatics, Systems and Communication (DISCo), University of Milan-Bicocca, Italy
giulia.bernardini@unimib.it

Roberto Grossi[2]
Department of Computer Science, University of Pisa, Italy and ERABLE Team, INRIA, France
grossi@di.unipi.it

Costas S. Iliopoulos
Department of Informatics, King's College London, UK
costas.iliopoulos@kcl.ac.uk

Nadia Pisanti[3]
Department of Computer Science, University of Pisa, Italy and ERABLE Team, INRIA, France
pisanti@di.unipi.it

Solon P. Pissis[4]
Department of Informatics, King's College London, UK
solon.pissis@kcl.ac.uk

Giovanna Rosone[5]
Department of Computer Science, University of Pisa, Italy
giovanna.rosone@unipi.it

Abstract

A generalised degenerate string (GD string) $\hat{S}$ is a sequence of n sets of strings of total size N, where the ith set contains strings of the same length k_i but this length can vary between different sets. We denote the sum of these lengths $k_0, k_1, \ldots, k_{n-1}$ by W. This type of uncertain sequence can represent, for example, a gapless multiple sequence alignment of width W in a compact form. Our first result in this paper is an $\mathcal{O}(N+M)$-time algorithm for deciding whether the intersection

[1] Partially supported by the project UNIPI PRA_2017_44 "Advanced computational methodologies for the analysis of biomedical data".
[2] Partially supported by the project UNIPI PRA_2017_44 "Advanced computational methodologies for the analysis of biomedical data".
[3] Partially supported by the project MIUR-SIR CMACBioSeq "Combinatorial methods for analysis and compression of biological sequences" grant n. RBSI146R5L and the project UNIPI PRA_2017_44 "Advanced computational methodologies for the analysis of biomedical data".
[4] Partially supported by the Royal Society project IE 161274 "Processing uncertain sequences: combinatorics and applications".
[5] Partially supported by the project MIUR-SIR CMACBioSeq "Combinatorial methods for analysis and compression of biological sequences" grant n. RBSI146R5L, the Royal Society project IE 161274 "Processing uncertain sequences: combinatorics and applications", and the project UNIPI PRA_2017_44 "Advanced computational methodologies for the analysis of biomedical data".

18th International Workshop on Algorithms in Bioinformatics (WABI 2018).
Editors: Laxmi Parida and Esko Ukkonen; Article No. 21; pp. 21:1–21:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of two GD strings of total sizes N and M, respectively, over an integer alphabet, is non-empty. This result is based on a combinatorial result of independent interest: although the intersection of two GD strings can be exponential in the total size of the two strings, it can be represented in only *linear* space. A similar result can be obtained by employing an automata-based approach but its cost is alphabet-dependent. We then apply our string comparison algorithm to compute palindromes in GD strings. We present an $\mathcal{O}(\min\{W, n^2\}N)$-time algorithm for computing all palindromes in $\hat{S}$. Furthermore, we show a similar conditional lower bound for computing maximal palindromes in $\hat{S}$. Finally, proof-of-concept experimental results are presented using real protein datasets.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases degenerate strings, generalised degenerate strings, elastic-degenerate strings, string comparison, palindromes

Digital Object Identifier 10.4230/LIPIcs.WABI.2018.21

1 Introduction

A *degenerate string* (or indeterminate string) over an alphabet Σ is a sequence of subsets of Σ. A great deal of research has been conducted on degenerate strings (see [1, 11, 20, 29, 32] and references therein). These types of uncertain sequences have been used extensively for flexible modelling of DNA sequences known as IUPAC-encoded DNA sequences [23].

In [19], the authors introduced a more general definition of degenerate strings: an *elastic-degenerate string* (ED string) $\tilde{S}$ over Σ is a sequence of subsets of Σ^* (see also network expressions [28]) with the aim of representing multiple genomic sequences [10]. That is, any set of $\tilde{S}$ does not contain, in general, only letters; a set may also contain strings, including the empty string. In a few recent papers on this notion, the authors provided several algorithms for pattern matching; specifically, for finding all exact [17] and approximate [8] occurrences of a standard string pattern in an ED text.

We introduce here another special type of uncertain sequence called generalised degenerate string; this can be viewed as an extension of degenerate strings or as a restricted variant of ED strings. Formally, a *generalised degenerate string* (GD string) $\hat{S}$ over Σ is a sequence of n sets of strings over Σ of total size N, where the ith set contains strings of the same length $k_i > 0$ but this length can vary between different sets. We denote the sum of these lengths $k_0, k_1, \ldots, k_{n-1}$ by W. Thus a GD string can be used to represent a *gapless* multiple sequence alignment (MSA) of fixed width, that is, for example, a high-scoring local alignment of multiple sequences, in a compact form; see Figure 1. This type of alignment is used for finding *functional* sequence elements [14]. For instance, searching for palindromic motifs in these type of alignments is an important problem since many transcription factors bind as homodimers to palindromes [26]. Specifically, a set of virus species can be clustered using high-scoring MSA to obtain subsets of viruses that have a *common* hairpin structure [27].

Our motivation for this paper comes from finding palindromes in these types of uncertain sequences. Let us start off with standard strings. A palindrome is a sequence that reads the same from left to right and from right to left. Detection of palindromic factors in texts is a classical and well-studied problem in algorithms on strings and combinatorics on words with a lot of variants arising out of different practical scenarios. In molecular biology, for instance, palindromic sequences are extensively studied: they are often distributed around promoters, introns, and untranslated regions, playing important roles in gene regulation and other cell

```
CA--AGCTCTATCTCGTA--TT
C---AGCCGAAGCTCGTATATT
CATCAAGTCAACGCAG----TT
```

(a) Multiple sequence alignment.

```
AGCTCTATCTCG
AGCCGAAGCTCG
AAGTCAACGCAG
```

(b) Local gapless alignment.

$$\hat{S} = \{\underline{\mathtt{A}}\} \cdot \begin{Bmatrix} \underline{\mathtt{GC}} \\ \mathtt{AG} \end{Bmatrix} \cdot \begin{Bmatrix} \mathtt{TCT} \\ \underline{\mathtt{CGA}} \\ \mathtt{TCA} \end{Bmatrix} \cdot \{\mathtt{A}\} \cdot \begin{Bmatrix} \mathtt{TCTC} \\ \underline{\mathtt{GCTC}} \\ \mathtt{CGCA} \end{Bmatrix} \cdot \{\underline{\mathtt{G}}\}$$

(c) GD string obtained from the local gapless alignment.

Figure 1 A GD string representing a gapless multiple sequence alignment.

processes (e.g. see [4]). In particular these are strings of the form $X\bar{X}^R$, also known as complemented palindromes, occurring in single-stranded DNA or, more commonly, in RNA, where X is a string and $\bar{X}^R$ is the reverse complement of X. In DNA, `C-G` are complements and `A-T` are complements; in RNA, `C-G` are complements and `A-U` are complements.

A string $X = X[0]X[1]\dots X[n-1]$ is said to have an initial palindrome of length k if its prefix of length k is a palindrome. Manacher first discovered an on-line algorithm that finds all initial palindromes in a string [25]. Later Apostolico et al observed that the algorithm given by Manacher is able to find all maximal palindromic factors in the string in $\mathcal{O}(n)$ time [6]. Gusfield gave an off-line linear-time algorithm to find all maximal palindromes in a string and also discussed the relation between biological sequences and gapped palindromes [18].

For uncertain sequences, we first need to have an algorithm for efficient *string comparison*, where automata provide the following baseline. Let $\hat{X}$ and $\hat{Y}$ be two GD (or two ED) strings of total sizes N and M, respectively. We first build the non-deterministic finite automaton (NFA) A of $\hat{X}$ and the NFA B of $\hat{Y}$ in time $\mathcal{O}(N+M)$. We then construct the product NFA C such that $L(C) = L(A) \cap L(B)$ in time $\mathcal{O}(NM)$. The non-emptiness decision problem, namely, checking if $L(C) \neq \emptyset$, is decidable in time linear in the size of C, using breadth-first search (BFS). Hence the comparison of $\hat{X}$ and $\hat{Y}$ can be done in time $\mathcal{O}(NM)$. It is known that if there existed faster methods for obtaining the automata intersection, then significant improvements would be implied to many long standing open problems [24]. Hence an immediate reduction to the problem of NFA intersection does not particularly help. For GD strings we show at the beginning of Section 3 that we can build an ad-hoc deterministic finite automaton (DFA) for $\hat{X}$ and $\hat{Y}$, so that the intersection can be performed efficiently, but this simple solution cannot achieve $\mathcal{O}(N+M)$ time as its cost is alphabet-dependent.

Our Contribution. Our first result in this paper is an $\mathcal{O}(N+M)$-time algorithm for deciding whether the intersection of two GD strings of sizes N and M, respectively, over an integer alphabet is non-empty. This result is based on a combinatorial result of independent interest: although the intersection of two GD strings can be exponential in the total size of the two strings, it can be represented in only linear space. An automata model of computation can also be employed to obtain these results but we present here an efficient implementation in the standard word RAM model with word size $w = \Omega(\log(N+M))$ that works also for integer alphabets. We then apply our string comparison tool to compute palindromes in GD strings. We present an $\mathcal{O}(\min\{W, n^2\}N)$-time algorithm for computing all palindromes in $\hat{S}$. Furthermore, we show a non-trivial $\Omega(n^2|\Sigma|)$ lower bound under the Strong Exponential Time Hypothesis [21, 22] for computing all maximal palindromes. Note that there exists an infinite

family of GD strings over an integer alphabet of size $|\Sigma| = \Theta(N)$ on which our algorithm requires time $\mathcal{O}(n^2 N)$ thus matching the conditional lower bound. Finally, proof-of-concept experimental results are presented using real protein datasets; specifically, on applying our tools to find the location of palindromes in immunoglobulins genes of the human V regions.

2 Preliminaries

An *alphabet* Σ is a non-empty finite set of letters of size $\sigma = |\Sigma|$. A *string* X on an alphabet Σ is a sequence of elements of Σ. The set of all strings on an alphabet Σ, including the *empty string* ε of length 0, is denoted by Σ^*. For any string X, we denote by $X[i \dots j]$ the *substring* or *factor* of X that *starts* at position i and *ends* at position j. In particular, $X[0 \dots j]$ is the *prefix* of X that ends at position j, and $X[i \dots |X| - 1]$ is the *suffix* of X that starts at position i, where $|X|$ denotes the *length* of X. The *suffix tree* of X (*generalised suffix tree* for a set of strings) is a compact trie representing all suffixes of X. We denote the *reversal* of X by string X^R, i.e. $X^R = X[|X| - 1]X[|X| - 2] \dots X[0]$.

A string P is said to be a *palindrome* if and only if $P = P^R$. If factor $X[i \dots j]$, $0 \leq i \leq j \leq n - 1$, of string X of length n is a palindrome, then $\frac{i+j}{2}$ is the *center* of $X[i \dots j]$ in X and $\frac{j-i+1}{2}$ is the *radius* of $X[i \dots j]$. In other words, a palindrome is a string that reads the same forward and backward, i.e. a string P is a palindrome if $P = YaY^R$ where Y is a string, Y^R is the reversal of Y and a is either a single letter or the empty string. Moreover, $X[i \dots j]$ is called a *palindromic factor* of X. It is said to be a *maximal palindrome* if there is no other palindrome in X with center $\frac{i+j}{2}$ and larger radius. Hence X has exactly $2n - 1$ maximal palindromes. A maximal palindrome P of X can be encoded as a pair (c, r), where c is the center of P in X and r is the radius of P.

▶ **Definition 1.** A *generalised degenerate string (GD string)* $\hat{S} = \hat{S}[0]\hat{S}[1] \dots \hat{S}[n - 1]$ of *length* n over an alphabet Σ is a finite sequence of n degenerate letters. Every *degenerate letter* $\hat{S}[i]$ of *width* $k_i > 0$, denoted also by $w(\hat{S}[i])$, is a finite non-empty set of strings $\hat{S}[i][j] \in \Sigma^{k_i}$, with $0 \leq j < |\hat{S}[i]|$. For any GD string $\hat{S}$, we denote by $\hat{S}[i] \dots \hat{S}[j]$ the *GD substring* of $\hat{S}$ that starts at position i and ends at position j.

▶ **Definition 2.** The *total size* N and *total width* W, denoted also by $w(\hat{S})$, of a GD string $\hat{S}$ are respectively defined as $N = \sum_{i=0}^{n-1} |\hat{S}[i]| \times k_i$ and $W = \sum_{i=0}^{n-1} k_i$.

In this work, we generally consider GD strings over an *integer alphabet* of size $\sigma = N^{\mathcal{O}(1)}$.

▶ **Example 3.** The GD string $\hat{S}$ of Figure 1(c) has length $n = 6$, size $N = 28$, and $W = 12$.

▶ **Definition 4.** Given two degenerate letters $\hat{X}$ and $\hat{Y}$, their *Cartesian concatenation* is

$$\hat{X} \otimes \hat{Y} = \{xy \mid x \in \hat{X}, y \in \hat{Y}\}.$$

When $\hat{Y} = \emptyset$ (resp. $\hat{X} = \emptyset$) we set $\hat{X} \otimes \hat{Y} = \hat{X}$ (resp. $= \hat{Y}$). Notice that $\otimes$ is associative.

▶ **Definition 5.** Consider a GD string $\hat{S}$ of length n. The *language of* $\hat{S}$ is

$$L(\hat{S}) = \hat{S}[0] \otimes \hat{S}[1] \otimes \cdots \otimes \hat{S}[n - 1].$$

Given two GD strings $\hat{R}$ and $\hat{S}$ of equal total width the *intersection* of their languages is defined by $L(\hat{R}) \cap L(\hat{S})$.

▶ **Definition 6.** Let $\hat{X} = \{ x_i \in \Sigma^k \}$ and $\hat{Y} = \{ y_j \in \Sigma^h \}$ be two degenerate letters on alphabet Σ. Further let us assume without loss of generality that $\hat{Y}$ is the set that contains the shorter strings (i.e. $h \leq k$). We define the *chop of* $\hat{X}$ *and* $\hat{Y}$ and the *active suffixes of* $\hat{X}$ *and* $\hat{Y}$ as follows:

- $\text{chop}_{\hat{X},\hat{Y}} = \{\, y_j \in \hat{Y} \mid y_j \text{ matches a prefix of } x_i \in \hat{X} \,\}$
- $\text{active}_{\hat{X},\hat{Y}} = \{\, x_i[h \dots k-1] \mid x_i[0 \dots h-1] \in \text{chop}_{\hat{X},\hat{Y}} \,\}$

Let $w(\text{chop}_{\hat{X},\hat{Y}}) = \min\{w(\hat{X}), w(\hat{Y})\}$. When $\text{active}_{\hat{X},\hat{Y}} = \{\varepsilon\}$, we set $\text{active}_{\hat{X},\hat{Y}} = \emptyset$. We then have that $\text{active}_{\hat{X},\hat{Y}} = \emptyset$ either if $h = k$ or if there is no match between any of the strings in $\hat{Y}$ and the prefix of a string in $\hat{X}$; i.e. $\text{chop}_{\hat{X},\hat{Y}} = \emptyset$.

▶ **Example 7.** Consider the following degenerate letters $\hat{X}$ and $\hat{Y}$ where $w(\hat{Y}) < w(\hat{X})$. The underlined strings in letter $\hat{Y}$ are prefixes of strings in letter $\hat{X}$, hence they are in $\text{chop}_{\hat{X},\hat{Y}}$. The suffixes of such strings in $\hat{X}$ are the active suffixes in $\text{active}_{\hat{X},\hat{Y}}$.

$$\hat{X} = \begin{Bmatrix} \underline{\texttt{TCC}}\texttt{TA} \\ \texttt{ATCGA} \\ \underline{\texttt{TCC}}\texttt{AC} \\ \underline{\texttt{CAT}}\texttt{TA} \end{Bmatrix} \quad \hat{Y} = \begin{Bmatrix} \texttt{GCA} \\ \underline{\texttt{CAT}} \\ \underline{\texttt{TCC}} \end{Bmatrix} \quad \text{chop}_{\hat{X},\hat{Y}} = \begin{Bmatrix} \texttt{CAT} \\ \texttt{TCC} \end{Bmatrix} \quad \text{active}_{\hat{X},\hat{Y}} = \begin{Bmatrix} \texttt{TA} \\ \texttt{AC} \end{Bmatrix}$$

▶ **Definition 8.** Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively. $\hat{R}[0] \dots \hat{R}[i]$ is the *prefix* of $\hat{R}$ that ends at position i. It is called *proper* if $i \neq r-1$. We say that $\hat{R}[0] \dots \hat{R}[i]$ is *synchronized* with $\hat{S}[0] \dots \hat{S}[j]$ if $w(\hat{R}[0] \dots \hat{R}[i]) = w(\hat{S}[0] \dots \hat{S}[j])$. We call these the *shortest synchronized prefixes* of $\hat{R}$ and $\hat{S}$, respectively, when $\forall\ i' < i, j' < j$ $w(\hat{R}[0] \dots \hat{R}[i']) \neq w(\hat{S}[0] \dots \hat{S}[j'])$.

3 GD String Comparison

In this section, we consider the fundamental problem of GD string comparison. Let $\hat{R}$ and $\hat{S}$ be of total size N and M, respectively. We provide an $\mathcal{O}(N+M)$-time algorithm in the standard word RAM model with word size $w = \Omega(\log(N+M))$ that works also for integer alphabets.

Before presenting our efficient implementation, we observe that there is the following simple algorithm based on DFAs. Each degenerate letter of $\hat{R}$ and $\hat{S}$ can be represented by a trie, where its leaves are collapsed to a single one. For every two consecutive degenerate letters, the collapsed leaves of the former trie coincide with the root of the latter trie. An acyclic DFA is obtained in this way, as illustrated in Appendix A. We can perform the comparison of $\hat{R}$ and $\hat{S}$ by intersecting their corresponding DFAs using BFS on their product DFA. The trivial upper bound on the number of reachable states is $\mathcal{O}(NM)$, but this can be improved to $\mathcal{O}(N+M)$ by exploiting the structure of the two input DFAs. Each state in such a DFA has a unique level: the common length of paths from the initial state; and this structure is *inherited* by the product DFA. In other words, a level-i state in the product DFA corresponds to a pair of level-i states in the input DFAs. Observe that a level-i state in one DFA is uniquely represented by the label of the path from the root of its trie, and for a fixed DFA and level, these labels have uniform lengths. Considering the two states composing a reachable state in the product DFA, it is easy to see that the shorter label must be a suffix of the longer label. Hence, the state in the DFA with longer labels at level i uniquely determines the state in the DFA with shorter labels at level i. Consequently, the number of reachable level-i states in the product DFA is bounded by the number of level-i states in the input DFAs, and the size is $\mathcal{O}(N+M)$.

We observe that the cost of implementing the above ideas has an extra logarithmic factor due to state branching and, moreover, GD string comparisons require to build the DFAs each time. We show how to obtain $\mathcal{O}(N+M)$ time for integer alphabets, *without* creating DFAs. We show that, even if the size of $L(\hat{R}) \cap L(\hat{S})$ can be exponential in the total sizes of $\hat{R}$ and $\hat{S}$ (Fact 9), the problem of GD string comparison, i.e. deciding whether $L(\hat{R}) \cap L(\hat{S})$ is non-empty, can be solved in time linear with respect to the sum of the total sizes of the two GD strings (Theorem 17) and is thus of independent interest.

▶ **Fact 9.** *Given two GD strings $\hat{R}$ and $\hat{S}$, $L(\hat{S}) \cap L(\hat{R})$ can have size exponential in the total sizes of $\hat{R}$ and $\hat{S}$.*

We next show when it is possible to factorize $L(\hat{R}) \cap L(\hat{S})$ into a Cartesian concatenation.

▶ **Lemma 10.** *Consider two GD strings $\hat{S} = \hat{S}'\hat{S}''$ and $\hat{R} = \hat{R}'\hat{R}''$ such that $w(\hat{S}) = w(\hat{R})$. If $\hat{S}'$ is synchronized with $\hat{R}'$, then $L(\hat{R}) \cap L(\hat{S}) = (L(\hat{R}') \cap L(\hat{S}')) \otimes (L(\hat{R}'') \cap L(\hat{S}''))$.*

Proof. It is clear that $L(\hat{S}) \cap L(\hat{R}) \supseteq (L(\hat{R}') \cap L(\hat{S}')) \otimes (L(\hat{S}'') \cap L(\hat{R}''))$. Indeed, consider a string $x \in L(\hat{R}') \cap L(\hat{S}')$ and a string $y \in L(\hat{S}'') \cap L(\hat{R}'')$: then, by the definition of Cartesian concatenation, $xy \in L(\hat{R}') \otimes L(\hat{R}'') = L(\hat{R})$ and $xy \in L(\hat{S}') \otimes L(\hat{S}'') = L(\hat{S})$.
We now prove the opposite inclusion. Consider a string $z \in L(\hat{S}) \cap L(\hat{R})$. By definition, $z = x_0x1 \dots x_{r-1} = y_0y_1 \dots y_{s-1}$, with $x_i \in \hat{R}[i], y_j \in \hat{S}[j], \forall\ 0 \le i \le r-1, \forall\ 0 \le j \le s-1$. Let $\hat{R}' = \hat{R}[0] \dots \hat{R}[\bar{i}]$, $\hat{S}' = \hat{S}[0] \dots \hat{S}[\bar{j}]$. Assume by contradiction that $z \notin (L(\hat{R}') \cap L(\hat{S}')) \otimes (L(\hat{S}'') \cap L(\hat{R}''))$: without loss of generality, $x_0 \dots x_{\bar{i}} \notin L(\hat{S}')$. Since $L(\hat{S}') \otimes L(\hat{S}'') = L(\hat{S})$, it follows that $z = x_0x_1 \dots x_{r-1} \notin L(\hat{S}) \implies z \notin L(\hat{S}) \cap L(\hat{R})$, that is a contradiction. ◀

By applying Lemma 10 wherever $\hat{R}$ and $\hat{S}$ have synchronized prefixes, we are then left with the problem of intersecting GD strings with no synchronized proper prefixes. We now define an alternative decomposition within such strings (see also Example 12).

▶ **Definition 11.** Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively, with no synchronized proper prefixes. We define

$$\text{c-chain}(\hat{R}, \hat{S}) = \max_q \{0 \le q \le r+s-2 \mid \text{chop}_q \neq \emptyset\},$$

where chop_i denotes the set $\text{chop}_{\hat{A}_i, \hat{B}_i}$, and $(\hat{A}_0, \hat{B}_0), (\hat{A}_1, \hat{B}_1), \dots, (\hat{A}_q, \hat{B}_q)$, $\text{pos}(\hat{A}_i), \text{pos}(\hat{B}_i)$ are recursively defined as follows:

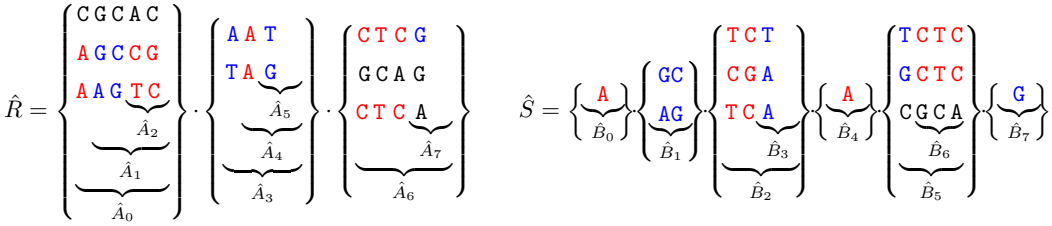
$\hat{A}_0 = \hat{R}[0], \hat{B}_0 = \hat{S}[0]$, and $\text{pos}(\hat{A}_0) = \text{pos}(\hat{B}_0) = 0$. For $0 < i \le r+s-2$, if $\text{chop}_{i-1} \neq \emptyset$,

$$\hat{A}_i = \begin{cases} \hat{R}[\text{pos}(\hat{A}_{i-1}) + 1] \text{ and } \text{pos}(\hat{A}_i) = \text{pos}(\hat{A}_{i-1}) + 1 & \text{if } w(\text{chop}_{i-1}) = w(\hat{A}_{i-1}) \\ \text{active}_{\hat{A}_{i-1}, \hat{B}_{i-1}} \text{ and } \text{pos}(\hat{A}_i) = \text{pos}(\hat{A}_{i-1}) & \text{otherwise} \end{cases}$$

$$\hat{B}_i = \begin{cases} \hat{S}[\text{pos}(\hat{B}_{i-1}) + 1] \text{ and } \text{pos}(\hat{B}_i) = \text{pos}(\hat{B}_{i-1}) + 1 & \text{if } w(\text{chop}_{i-1}) = w(\hat{B}_{i-1}) \\ \text{active}_{\hat{A}_{i-1}, \hat{B}_{i-1}} \text{ and } \text{pos}(\hat{B}_i) = \text{pos}(\hat{B}_{i-1}) & \text{otherwise} \end{cases}$$

The generation of pairs $(\hat{A}_i, \hat{B}_i)$ stops at $i = q$ either if $q = r+s-2$, or when $\text{chop}_{q+1} = \emptyset$, in which case $\hat{R}$ and $\hat{S}$ only match until $(\hat{A}_q, \hat{B}_q)$. Intuitively, $\hat{A}_i$ (respectively, $\hat{B}_i$) represents suffixes of the current position of $\hat{R}$ (respectively, of $\hat{S}$), while $\text{pos}(\hat{B}_i)$ (respectively, $\text{pos}(\hat{A}_i)$) tells *which* position of $\hat{R}$ (respectively, $\hat{S}$) we are chopping.

▶ **Example 12** (Definition 11)**.** Consider the following GD strings $\hat{R}$ and $\hat{S}$ with no synchronized proper prefixes: chop_0 is the first red set from the left, chop_1 is the first blue one, chop_2 is the second red one, etc. The c-chain$(\hat{R}, \hat{S})$ terminates when $q = 7$.



▶ **Definition 13.** Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively, with $w(\hat{R}) = w(\hat{S})$ and no synchronized proper prefixes. We define $G_{\hat{R},\hat{S}}$ as a directed acyclic graph with a structure of up to $r + s - 1$ levels, each node being a set of strings, as follows, where we assume without loss of generality that $w(\hat{R}[0]) > w(\hat{S}[0])$:

Level $k = 0$: consists of a single node:

$n_0 = \{x \in \hat{R}[0] \mid x = y_0 \dots y_{q_0}$ with $y_j \in \text{chop}_j \, \forall j : 0 \le j \le q_0\}$, where q_0 is the index of the rightmost chop containing suffixes of $\hat{R}[0]$.

Level $k > 0$: consists of $\ell = |\text{chop}_{q_{k-1}}|$ nodes. Assuming without loss of generality that level $k-1$ has been built with suffixes of $\hat{R}[\text{pos}(\hat{A}_{q_{k-1}})]$, level k contains suffixes of a position of $\hat{S}$. Let $c_0, \dots, c_{\ell-1}$ denote the elements of $\text{chop}_{q_{k-1}}$. Then, for $0 \le i \le \ell-1$, the i-th node of level k is:

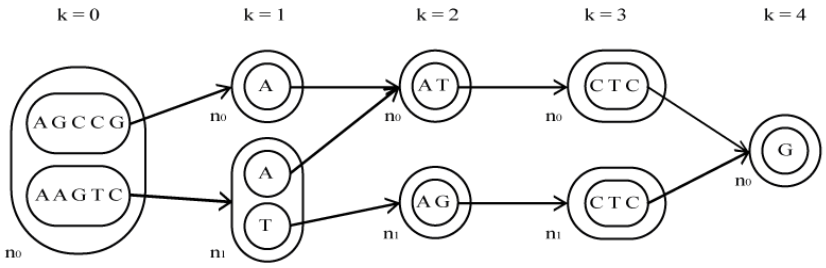
$n_i = \{y_{q_{k-1}+1} \dots y_{q_k} \mid c_i y_{q_{k-1}+1} \dots y_{q_k} \in \hat{B}_{q_{k-1}}$ with $y_j \in \text{chop}_j \, \forall j : q_{k-1}+1 \le j \le q_k\}$, where q_k is the index of the rightmost chop containing suffixes of $\hat{S}[\text{pos}(\hat{B}_{q_{k-1}})]$.

Every string in level $k-1$ whose suffix is c_i is the source of an edge having the whole node n_i as a sink.

We define $\text{paths}(G_{\hat{R},\hat{S}})$ as the set of strings spelled by a path in $G_{\hat{R},\hat{S}}$ that starts at n_0 and ends at the last level.

Note that the size of $G_{\hat{R},\hat{S}}$ is at most linear in the sum of the sizes of $\hat{R}$ and $\hat{S}$, as the nodes contain strings either in $\hat{R}$ or in $\hat{S}$ with no duplications, and each node has out-degree equal to the number of strings it contains.

▶ **Example 14** (Definition 13). $G_{\hat{R},\hat{S}}$ for the GD strings $\hat{R}, \hat{S}$ of Example 12 is:



$q_0 = 2$ and the strings in level 0 belong to $(\text{chop}_0 \otimes \text{chop}_1 \otimes \text{chop}_2) \cap \hat{R}[0]$. Level 1 contains suffixes of strings in $\hat{B}_2$ (and of strings in $\hat{B}_3$ as $\text{chop}_3 = \{\texttt{A}, \texttt{T}\}$ and indeed $q_1 = 3$), level 2 suffixes of strings in $\hat{A}_3$ (as $q_2 = 5$), level 3 suffixes of strings in $\hat{B}_5$ ($q_3 = 6$), level 4 suffixes of strings in $\hat{A}_6$ ($q_4 = 7$). The three paths from level 0 to level 4 correspond to the three strings in $L(\hat{R}) \cap L(\hat{S})$: `AGCCGAATCTCG, AAGTCAATCTCG, AAGTCTAGCTCG`.

Let $G^k_{\hat{R},\hat{S}}$ be $G_{\hat{R},\hat{S}}$ truncated at level k, and let $|G^k_{\hat{R},\hat{S}}|$ be the length of the strings it spells. Let $L_k(\hat{S})$ denote the set of prefixes of length $|G^k_{\hat{R},\hat{S}}|$ of $L(\hat{S})$.

▶ **Lemma 15.** *Let $\hat{R}, \hat{S}$ be two GD strings with $w(\hat{R}) = w(\hat{S}) = W$ and no synchronized proper prefixes. Then $L_k(\hat{S}) \cap L_k(\hat{R}) = \text{paths}(G^k_{\hat{R},\hat{S}})$ for all levels k of $G_{\hat{R},\hat{S}}$ such that $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$.*

Proof. Again, let us assume without loss of generality that $w(\hat{R}[0]) > w(\hat{S}[0])$. We prove the result by induction on k.

[Level $k = 0$] By construction, n_0 contains strings in $\hat{R}[0] \cap (\text{chop}_0 \otimes \cdots \otimes \text{chop}_{q_0})$, which have length $|G^0_{\hat{R},\hat{S}}|$, and are also in $\hat{S}[0]$, and hence belong to both $L_0(\hat{S})$ and $L_0(\hat{R})$.

[Level $k > 0$] By inductive hypothesis, we have that $L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R}) = \text{paths}(G^{k-1}_{\hat{R},\hat{S}})$: suppose that $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$, otherwise the graph ends at level $k-1$. We first show that $\text{paths}(G^k_{\hat{R},\hat{S}}) \subseteq L_k(\hat{S}) \cap L_k(\hat{R})$: by Definition 13, any $z \in \text{paths}(G^k_{\hat{R},\hat{S}})$ can be written as $z = z'z''$ with z' in $\text{paths}(G^{k-1}_{\hat{R},\hat{S}})$ and with z'' that belongs to some node at level k of $G^k_{\hat{R},\hat{S}}$ reached by an edge leaving a suffix of z'. By inductive hypothesis $z' \in L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R})$ and, again by Definition 13, $z'' \in \text{chop}_{q_{k-1}+1} \otimes \cdots \otimes \text{chop}_{q_k}$; since $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$ these chops are not empty, their concatenation contains the suffix of length $|G^k_{\hat{R},\hat{S}}| - |G^{k-1}_{\hat{R},\hat{S}}|$ of strings in both $L_k(\hat{R})$ and $L_k(\hat{S})$, and hence $z \in L_k(\hat{S}) \cap L_k(\hat{R})$.
We now show that $L_k(\hat{S}) \cap L_k(\hat{R}) \subseteq \text{paths}(G^k_{\hat{R},\hat{S}})$: consider string $u \in L_k(\hat{S}) \cap L_k(\hat{R})$ that can be written as $u = u'u''$ with u' the prefix of u having length $|G^{k-1}_{\hat{R},\hat{S}}|$ which then belongs to $L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R})$; then, by inductive hypothesis, $u' \in \text{paths}(G^{k-1}_{\hat{R},\hat{S}})$ and, since $u \in L_k(\hat{S}) \cap L_k(\hat{R})$, then there is an edge linking a suffix of u' at level $k-1$ with a node at level k of $G^k_{\hat{R},\hat{S}}$ containing a $|G^k_{\hat{R},\hat{S}}| - |G^{k-1}_{\hat{R},\hat{S}}|$ long suffix u'' of u, and hence $u \in \text{paths}(G^k_{\hat{R},\hat{S}})$. ◀

As a special case of Lemma 15, if $L(\hat{S}) \cap L(\hat{R}) \neq \emptyset$, then $G_{\hat{R},\hat{S}}$ is built up to the last level and the following holds.

▶ **Theorem 16.** *Let $\hat{R}, \hat{S}$ be two GD strings having lengths, respectively, r and s, with $w(\hat{R}) = w(\hat{S})$ and no synchronized proper prefixes. Then $G_{\hat{R},\hat{S}}$ has exactly $r + s - 1$ levels, and we have that $L(\hat{S}) \cap L(\hat{R}) = \text{paths}(G_{\hat{R},\hat{S}})$.*

$G_{\hat{R},\hat{S}}$ is thus a linear-sized representation of the possibly exponential-sized (Fact 9) set $L(\hat{S}) \cap L(\hat{R})$.

We now show an $\mathcal{O}(N + M)$-time algorithm for the standard word RAM model, denoted by GDSC, that decides whether $L(\hat{R})$ and $L(\hat{S})$ share at least one string (returns 1) or not (returns 0). GDSC starts with constructing the generalized suffix tree $T_{\hat{R},\hat{S}}$ of all the strings in $\hat{R}$ and $\hat{S}$. Then it scans $\hat{R}$ and $\hat{S}$ starting with $\hat{R}[0]$ and $\hat{S}[0]$ storing in $\text{chop}_{\hat{R},\hat{S}}$ the latest chop_i and in $\text{active}_{\hat{R},\hat{S}}$ the latest $\text{active}_{\hat{A}_i,\hat{B}_i}$ using $T_{\hat{R},\hat{S}}$. For an efficient implementation, suffixes in $\text{active}_{\hat{R},\hat{S}}$ are stored (e.g. for $\text{active}_{\hat{A}_0,\hat{B}_0}$ assuming that $w(\hat{R}[0]) > w(\hat{S}[0])$) as index positions of $\hat{R}[0]$ and the starting position of the suffix as $\text{active}_{\hat{R},\hat{S}}$.suff. The next comparison is made between the corresponding suffixes of $\hat{R}[0]$ of length $w(\hat{R}[0]) - \text{active}_{\hat{R},\hat{S}}$.suff and $\hat{S}[1]$, identifying first the minimum length of the two, and proceeding with the same process. The comparison of letters can be: (i) between $\hat{R}[i]$ and $\hat{S}[j]$; or (ii) between the corresponding strings of $\text{active}_{\hat{R},\hat{S}}$.index and $\hat{R}[i]$; or (iii) between the corresponding strings of $\text{active}_{\hat{R},\hat{S}}$.index and $\hat{S}[j]$. If the two GD strings have a synchronized proper prefix, this will result in $\text{active}_{\hat{R},\hat{S}} = \emptyset$ at positions i in $\hat{R}$ and j in $\hat{S}$. At this point, the comparison is restarted with the immediately following pair of degenerate letters.

▶ **Theorem 17.** *Algorithm* GDSC *is correct. Given two GD strings $\hat{R}$ and $\hat{S}$ of total sizes N and M, respectively, over an integer alphabet, algorithm* GDSC *requires $\mathcal{O}(N + M)$ time.*

Proof. The correctness follows directly from Lemma 10, Lemma 15, and Theorem 16.

Constructing the generalized suffix tree $T_{\hat{R},\hat{S}}$ can be done in time $\mathcal{O}(N + M)$ [12]. For the sets pair $(\hat{A}_i, \hat{B}_i)$ as in Definition 11, such that $w(\hat{A}_i) = k$ and $w(\hat{A}_i) \leq w(\hat{B}_i)$, we query $T_{\hat{R},\hat{S}}$ with the k-length prefixes of strings in $\hat{B}_i$. For integer alphabets, instead of spelling the strings from the root of $T_{\hat{R},\hat{S}}$, we locate the corresponding terminal nodes for $(\hat{A}_i, \hat{B}_i)$. It then suffices to find longest common prefixes between these suffixes to simulate the querying

process. Since all suffixes are lexicographically sorted during the construction of $T_{\hat{R},\hat{S}}$, we can also have the suffixes considered by pair $(\hat{A}_i, \hat{B}_i)$ lexicographically ranked with respect to $(\hat{A}_i, \hat{B}_i)$. Hence we do not perform the longest common prefix operation for all possible suffix pairs, but only for the lexicographically adjacent ones within this group. This can be done in $\mathcal{O}(1)$ time per pair after $\mathcal{O}(N+M)$-time pre-processing over $T_{\hat{R},\hat{S}}$ [7]. chop_i is thus populated with the k-length prefixes of strings in $\hat{B}_i$ found in $\hat{A}_i$. The set $\text{active}_{\hat{A}_i,\hat{B}_i}$ of active suffixes can be found by chopping the suffixes of the string in $\hat{B}_i$ from their prefixes successfully queried in $T_{\hat{R},\hat{S}}$. This requires time $\mathcal{O}(|\hat{A}_i| + |\hat{B}_i|)$ for processing $(\hat{A}_i, \hat{B}_i)$.

Let $\hat{R}$ and $\hat{S}$ be of length r and s, respectively. Assume that $\hat{R}$ and $\hat{S}$ have no synchronized proper prefixes. Then Theorem 16 ensures that the total number of comparisons cannot exceed $r+s-2$: this results in a time complexity of $\mathcal{O}(N+M+\sum_{i=0}^{r+s-2}(|\hat{A}_i|+|\hat{B}_i|)) = \mathcal{O}(N+M)$.

If $\hat{R}$ and $\hat{S}$ have synchronized proper prefixes, we perform the comparison up to the shortest synchronized prefixes (i.e. the set of active suffixes becomes empty) and then restart the procedure from the immediately following pair of degenerate letters. Clearly the total number of comparisons also in this case cannot be more than $r+s-2$. ◀

4 Computing Palindromes in GD Strings

Armed with the efficient GD string comparison tool, we shift our focus on our initial motivation, namely, computing palindromes in GD strings.

▶ **Definition 18.** A GD string $\hat{S}$ is a *GD palindrome* if there exists a string in $L(\hat{S})$ that is a palindrome.

A GD palindrome $\hat{S}[i]\dots\hat{S}[j]$ in $\hat{S}$, whose total width is $w(\hat{S}[i]\dots\hat{S}[j])$, can be encoded as a pair (c,r), where its *center* is $c = \frac{w(\hat{S}[0]\dots\hat{S}[i-1])+w(\hat{S}[0]\dots\hat{S}[j])-1}{2}$, when $i>0$, otherwise, $c = \frac{w(\hat{S}[0]\dots\hat{S}[j])-1}{2}$, when $i=0$; its *radius* is $r = \frac{w(\hat{S}[i]\dots\hat{S}[j])}{2}$. $\hat{S}[i]\dots\hat{S}[j]$ is called *maximal* if no other GD palindrome (c,r') exists in $\hat{S}$ with $r'>r$. Note that we only consider the GD palindromes $\hat{S}[i]\dots\hat{S}[j]$ that start with the first letter of some string $X \in \hat{S}[i]$ and end with the last letter of some string $Y \in \hat{S}[j]$, while the center can be anywhere: in between or inside degenerate letters. That is, in $\hat{S}$ there are $2\cdot w(\hat{S}) - 1 = 2W-1$ possible centers.

▶ **Example 19.** Consider the GD string $\hat{S}$ of Figure 1(c) where palindromes are underlined; one starts at $\hat{S}[0]$ and ends at $\hat{S}[2]$: it corresponds to $(c,r) = (2.5, 3)$. A second palindrome starts at $\hat{S}[4]$ and ends at $\hat{S}[5]$: it corresponds to $(c,r) = (9, 2.5)$.

In this section, we consider the following problem. Given a GD string $\hat{S}$ of length n, total size N, and total width W, find all GD strings $\hat{S}[i]\dots\hat{S}[j]$, with $0 \le i \le j \le n-1$, that are GD palindromes. We give two alternative algorithms: one finds all GD palindromes seeking them for all (i,j) pairs; and the other one finds them starting from all possible centers. The two algorithms have different time complexities: which one is faster depends on W, N, and n. In fact, they compute all GD palindromes, but report only the maximal ones.

We first describe algorithm MaxPalPairs. For all i,j positions within $\hat{S}$, in order to check whether $\hat{S}[i]\dots\hat{S}[j]$ is a GD palindrome, we apply the GDSC algorithm to $\hat{S}[i]\dots\hat{S}[j]$ and its reverse, denoted by $rev(\hat{S}[i]\dots\hat{S}[j])$; the reverse is defined by reversing the sequence of degenerate letters and also reversing the strings in every degenerate letter. GD palindromes are, finally, sorted per center, and the maximal GD palindromes are reported. Sorting the (i,j) pairs by their centers can be done in $\mathcal{O}(W)$ time using bucket sort, which is bounded by $\mathcal{O}(N)$ since $N \ge W$.

Since there are $\mathcal{O}(n^2)$ pairs (i, j), and since by Theorem 17 algorithm GDSC takes time proportional to the total size of $\hat{S}[i] \dots \hat{S}[j]$ to check whether $\hat{S}[i] \dots \hat{S}[j]$ is a GD palindrome, algorithm MaxPalPairs takes $\mathcal{O}(n^2 N)$ time in total. In algorithm MaxPalCenters, we consider all possible centers c of $\hat{S}$. In the case when c is in between two degenerate letters we simply try to extend to the left and to the right via applying GDSC. In the case when c is inside a degenerate letter we intuitively split the letter vertically into two letters and try to extend to the left and to the right via applying GDSC. At each extension step of this procedure we maintain two GD strings $\hat{L}$ (left of the center) and $\hat{R}$ (right of the center) such that they are of the same total width. We consider the reverse of $\hat{L}$ (similar to algorithm MaxPalPairs) for the comparison. In the case where c occurs inside a degenerate letter to make sure we do not identify palindromes which do not exist, for all j split strings of the degenerate letter, we check that $\hat{L}^R[0][j][0 \dots k-1] = \hat{R}[0][j][0 \dots k-1]$ where $\hat{L}^R = rev(\hat{L})$ and $k = \min(w(L^R[0]), w(\hat{R}[0]))$. If no matches are found, we move onto the next center. Otherwise, when a match is found, we update $rev(\hat{L})$ and $\hat{R}$ with the remainder of the split degenerate letter (if its length is greater than k), as well as the next degenerate letters. Algorithm GDSC is applied to compare $rev(\hat{L})$ and $\hat{R}$. After a positive comparison, we overwrite $\hat{L}$ and $\hat{R}$ by adding the degenerate letters of the current extension until $w(\hat{L}) = w(\hat{R})$ (or until the end of the string is reached). This process is repeated as long as GDSC returns a positive comparison, that is, until the maximal GD palindrome with center c is found. The radius reported is then the total sum of all values of $w(\hat{L})$. If GDSC returns a negative comparison at center c, we proceed with the next center, because we clearly cannot have a GD palindrome centered at c extended further if $rev(\hat{L}) \cap \hat{R}$ is empty.

By Theorem 17 and the fact that there are $2W - 1$ possible centers, we have that algorithm MaxPalCenters takes $\mathcal{O}(WN)$ time in total. We obtain the following result.

▶ **Theorem 20.** *Given a GD string of length n, total size N, and total width W, over an integer alphabet, all (maximal) GD palindromes can be computed in time $\mathcal{O}(\min\{W, n^2\}N)$.*

The problem that gained significant attention recently is the factorization of a string X of length n into a sequence of palindromes [3, 13, 30, 9, 5, 2]. We say that $X_1, X_2, \dots, X_\ell$ is a (maximal) palindromic factorization of string X, if every X_i is a (maximal) palindrome, $X = X_1 X_2 \dots X_\ell$, and ℓ is minimal. In biological applications we need to factorize a sequence into palindromes in order to identify *hairpins*, patterns that occur in single-stranded DNA or, more commonly, in RNA. Next, we define and solve the same problem for GD strings.

▶ **Definition 21.** A *(maximal) GD palindromic factorization* of a GD string $\hat{S}$ is a sequence $\hat{P}_1, \dots, \hat{P}_\ell$ of GD strings, such that: (i) every $\hat{P}_i$ is either a (maximal) GD palindrome or a degenerate letter of $\hat{S}$; (ii) $\hat{S} = \hat{P}_1 \dots \hat{P}_\ell$; (iii) ℓ is minimal.

After locating all (maximal) GD palindromes in $\hat{S}$ using Theorem 20, we are in a position to amend the algorithm of Alatabbi et al [3] to find a (maximal) GD palindromic factorization of $\hat{S}$. We define a directed graph $\mathcal{G}_{\hat{S}} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{i \mid 0 \leq i \leq n\}$ and $\mathcal{E} = \{(i, j+1) \mid \hat{S}[i \dots j]$ (maximal) GD palindrome of $\hat{S}\} \cup \{(i, i+1) | 0 \leq i < n\}$. Note that $\mathcal{V}$ contains a node n being the sink of edges representing (maximal) GD palindromes ending at $\hat{S}[n-1]$. For maximal GD palindromes, $\mathcal{E}$ contains no more than $3W$ edges, as the maximum number of maximal GD palindromes is $2W - 1$. For GD palindromes, $\mathcal{E}$ contains $\mathcal{O}(n^2)$ edges, as the maximum number of GD palindromes is $\mathcal{O}(n^2)$. A shortest path in $\mathcal{G}_{\hat{S}}$ from 0 to n gives a (maximal) GD palindromic factorization. For maximal GD palindromes, the size of $\mathcal{G}_{\hat{S}}$ is $\mathcal{O}(W)$, as $n \leq W$, and so finding this shortest path requires $\mathcal{O}(W)$ time using a standard algorithm. For GD palindromes, the size of $\mathcal{G}_{\hat{S}}$, and thus the time, is $\mathcal{O}(n^2)$.

▶ **Theorem 22.** *Given a GD string $\hat{S}$ of length n, total size N, and total width W, over an integer alphabet, a (maximal) GD palindromic factorization of $\hat{S}$ can be computed in time $\mathcal{O}(\min\{W, n^2\}N)$.*

5 A Conditional Lower Bound under SETH

In this section, we show a conditional lower bound for computing palindromes in degenerate strings. Let us first define the 2-Orthogonal Vectors problem. Given two sets $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$ of d-bit vectors, where $d = \omega(\log n)$, the *2-Orthogonal Vectors* problem asks the following question: is there any pair α_i, β_j of vectors that is orthogonal? Namely, is $\sum_{k=0}^{d-1} \alpha_i[k] \cdot \beta_j[k]$ equal to 0? For the moderate dimension of this problem, we follow [16], assuming $n^{2-\epsilon}d^{\mathcal{O}(1)} \leq n^2 d$. The following result is known.

▶ **Theorem 23** ([16, 21, 22, 33])**.** *The 2-Orthogonal Vectors problem cannot be solved in $\mathcal{O}(n^{2-\epsilon} \cdot d^{\mathcal{O}(1)})$ time, for any $\epsilon > 0$, unless the* Strong Exponential Time Hypothesis *fails.*

We next show that the 2-Orthogonal Vectors problem can be reduced to computing maximal palindromes in degenerate strings thus obtaining a similar conditional lower bound to the upper bound obtained in Theorem 20 for computing all GD palindromes.

▶ **Theorem 24.** *Given a degenerate string of length $4n$ over an alphabet of size $\sigma = \omega(\log n)$, all maximal GD palindromes cannot be computed in $\mathcal{O}(n^{2-\epsilon} \cdot \sigma^{\mathcal{O}(1)})$ time, for any $\epsilon > 0$, unless the* Strong Exponential Time Hypothesis *fails.*

Proof. Let $d = \sigma$ and consider the alphabet $\Sigma = \{0, 1, \ldots, \sigma - 1\}$. We say that two subsets of Σ *match* if they have a common element. Given a d-bit vector α, we define $\mu(\alpha)$ to be the following subset of Σ: $s \in \mu(\alpha)$ if and only if $\alpha[s] = 1$. Thus, two vectors α and β are orthogonal if and only if the sets $\mu(\alpha)$ and $\mu(\beta)$ are disjoint. In the string comparison setting, two degenerate letters $\mu(\alpha)$ and $\mu(\beta)$ *do not match* if and only if α and β are orthogonal. The reduction works as follows. Given $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$, we construct the following simple degenerate string of length $4n$ in time $\mathcal{O}(n\sigma)$:

$$S = \underbrace{\mu(\alpha_1)\mu(\beta_1)\mu(\alpha_2)\mu(\beta_2)\ldots\mu(\alpha_n)\mu(\beta_n)}\,\mu(\alpha_1)\mu(\beta_1)\mu(\alpha_2)\mu(\beta_2)\ldots\mu(\alpha_n)\mu(\beta_n).$$

$$\ldots \qquad\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}$$

Then the 2-Orthogonal Vectors problem for the sets A and B has a positive answer if and only if at any position of S, from 0 to $2n$, there *does not occur* a palindrome of length at least $2n$. All such occurrences can be easily verified from the respective palindrome centers in time $\mathcal{O}(n)$. In other words, if at any position of S there does not occur a palindrome of length at least $2n$, this is because we have a mismatch between a pair $\mu(\alpha_i), \mu(\beta_j)$ of letters, which implies that there exists a pair α_i, β_j of orthogonal vectors. Also, by the construction, all such pairs are to be (implicitly) compared, and thus, if there exists any pair that is orthogonal the corresponding mismatch will result in a palindrome of length less than $2n$. ◀

6 Experimental Results

We present here a proof-of-concept experiment but we anticipate that the algorithmic tools developed in this paper are applicable in a wide range of biological applications.

We first obtained the amino acid sequences of 5 immunoglobulins within the human V regions [15] and converted these into mRNA sequences [31]. The letters `X`, `S`, `T`, `Y`, `Z`, `R`

Table 1 Coordinates of (maximal) palindromes identified within hypervariable regions I and II.

	Hypervariable Region			
	I		II	
V	[34]	**This paper**	[34]	**This paper**
V_kII	18-27	11-36	119-130	118-131
	104-113	104-113	169-180	169-180
V_kIII	18-27	11-30	132-142	131-145
V_λII	63-74	62-81	140-152	140-152
V_λIII	51-74	50-75	132-143	131-144
V_λV	96-104	95-104	134-141	134-141

and `H` were replaced by degenerate letters according to IUPAC [23]. Each other letter, $c \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{U}\}$, was treated as a single degenerate letter $\{c\}$. An average of 47% of the total number of positions within the 5 sequences consisted of one of the following: `X`, `S`, `T`, `Y`, `Z`, `R` and `H`. We then used algorithm MaxPalPairs to find all maximal palindromes in the 5 sequences. Table 1 shows the palindromes identified within hypervariable regions I and II. Our results are in accordance with Wuilmart et al [34] who presented a *statistical* (fundamentally different) method to identify the location of palindromes within regions of immunoglobulin genes. The ranges we report are greater than or equal to the ones of [34] due to the *maximality* criterion.

References

1 Karl Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

2 Michał Adamczyk, Mai Alzamel, Panagiotis Charalampopoulos, Costas S. Iliopoulos, and Jakub Radoszewski. Palindromic decompositions with gaps and errors. In *CSR*, volume 10304 of *LNCS*, pages 48–61. Springer International Publishing, 2017.

3 Ali Alatabbi, Costas S. Iliopoulos, and M. Sohel Rahman. Maximal palindromic factorization. In *PSC*, pages 70–77, 2013.

4 Yannis Almirantis, Panagiotis Charalampopoulos, Jia Gao, Costas S. Iliopoulos, Manal Mohamed, Solon P. Pissis, and Dimitris Polychronopoulos. On avoided words, absent words, and their application to biological sequence analysis. *Algorithms for Molecular Biology*, 12(1):5, 2017.

5 Mai Alzamel, Jia Gao, Costas S. Iliopoulos, Chang Liu, and Solon P. Pissis. Efficient computation of palindromes in sequences with uncertainties. In *EANN*, volume 744 of *CCIS*, pages 620–629. Springer, 2017.

6 Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 141(1):163–173, 1995.

7 Michael A. Bender and Martín Farach-Colton. The LCA problem revisited. In *LATIN*, volume 1776 of *LNCS*, pages 88–94. Springer, 2000.

8 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Pattern matching on elastic-degenerate text with errors. In *SPIRE*, volume 10508 of *LNCS*, pages 74–90. Springer, 2017.

9 Kirill Borozdin, Dmitry Kosolobov, Mikhail Rubinchik, and Arseny M. Shur. Palindromic Length in Linear Time. In *CPM*, volume 78 of *LIPIcs*, pages 23:1–23:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

10 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, pages 1–18, 2016.

11 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Covering problems for partial words and for indeterminate strings. *Theoretical Computer Science*, 698:25–39, 2017.

12 Martin Farach. Optimal suffix tree construction with large alphabets. In *FOCS*, pages 137–143. IEEE, 1997.

13 Gabriele Fici, Travis Gagie, Juha Kärkkäinen, and Dominik Kempa. A subquadratic algorithm for minimum palindromic factorization. *Journal of Discrete Algorithms*, 28:41–48, 2014.

14 Martin C. Frith, Ulla Hansen, John L. Spouge, and Zhiping Weng. Finding functional sequence elements by multiple local alignment. *Nucleic Acids Res.*, 32(1):189–200, 2004.

15 J. A. Gally and G. M. Edelman. The genetic control of immunoglobulin synthesis. *Annual Review of Genetics*, 6(1):1–46, 1972.

16 Jiawei Gao and Russell Impagliazzo. Orthogonal vectors is hard for first-order properties on sparse graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:53, 2016.

17 Roberto Grossi, Costas S. Iliopoulos, Chang Liu, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, Giovanna Rosone, Fatima Vayani, and Luca Versari. On-line pattern matching on a set of similar texts. In *CPM*, LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

18 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.

19 Costas S. Iliopoulos, Ritu Kundu, and Solon P. Pissis. Efficient pattern matching in elastic-degenerate texts. In *LATA*, volume 10168 of *LNCS*, pages 131–142. Springer International Publishing, 2017.

20 Costas S. Iliopoulos and Jakub Radoszewski. Truly Subquadratic-Time Extension Queries and Periodicity Detection in Strings with Uncertainties. In *CPM*, volume 54 of *LIPIcs*, pages 8:1–8:12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

21 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

22 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

23 IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20):4022–4027, 1970.

24 Richard J. Lipton. *On The Intersection of Finite Automata*, pages 145–148. Springer US, Boston, MA, 2010.

25 Glenn Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *Journal of the ACM*, 22(3):346–351, 1975.

26 Lee Ann McCue, William Thompson, Steven Carmack, Michael P. Ryan, Jun S. Liu, Victoria Derbyshire, and Charles E. Lawrence. Phylogenetic footprinting of transcription factor binding sites in proteobacterial genomes. *Nucleic Acids Res.*, 29(3):774–782, 2001.

27 Brejnev Muhizi Muhire, Michael Golden, Ben Murrell, Pierre Lefeuvre, Jean-Michel Lett, Alistair Gray, Art YF Poon, Nobubelo Kwanele Ngandu, Yves Semegni, Emil Pavlov Tanov, et al. Evidence of pervasive biologically functional secondary structures within the genomes of eukaryotic single-stranded DNA viruses. *Journal of virology*, 88(4):1972–1989, 2014.

28 Eugene W Myers. Approximate matching of network expressions with spacers. *Journal of Computational Biology*, 3(1):33–51, 1996.

29 Nadia Pisanti, Henry Soldano, Mathilde Carpentier, and Joël Pothier. A relational extension of the notion of motifs: Application to the common 3d protein substructures searching problem. *Journal of Computational Biology*, 16(12):1635–1660, 2009.

30 Mikhail Rubinchik and Arseny M. Shur. Eertree: An efficient data structure for processing palindromes in strings. In *IWOCA*, volume 9538 of *LNCS*, pages 321–333. Springer International Publishing, 2016.

31 Randall T. Schuh. Major patterns in vertebrate evolution. *Systematic Biology*, 27(2):172, 1978.

32 Henry Soldano, Alain Viari, and Marc Champesme. Searching for flexible repeated patterns using a non-transitive similarity relation. *Pattern Recognition Letters*, 16(3):233–246, 1995.

33 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci*, 348(2-3):357–365, 2005.

34 C. Wuilmart, J. Urbain, and D. Givol. On the location of palindromes in immunoglobulin genes. *Proceedings of the National Academy of Sciences of the United States of America*, 74(6):2526–2530, 1977.

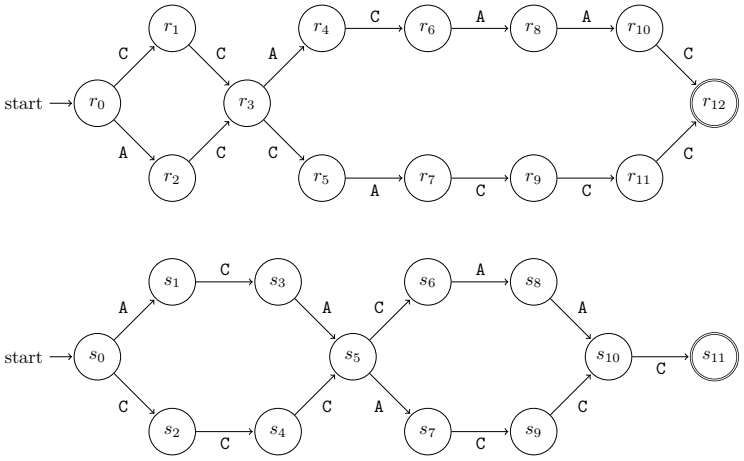
APPENDIX

A GD String Comparison Using Automata

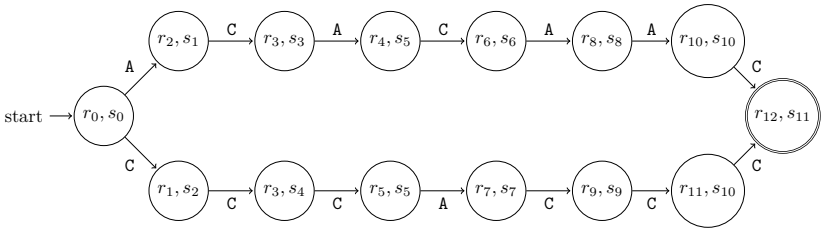
▶ **Example 25.** We illustrate here a simple automata-based approach. Say we want to compare the following two GD strings:

$$\hat{R} = \begin{Bmatrix} \texttt{AC} \\ \texttt{CC} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{ACAAC} \\ \texttt{CACCC} \end{Bmatrix} \qquad \hat{S} = \begin{Bmatrix} \texttt{ACA} \\ \texttt{CCC} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{ACC} \\ \texttt{CAA} \end{Bmatrix} \cdot \{\texttt{C}\}\,.$$

We construct the DFA for $\hat{R}$ and the DFA for $\hat{S}$.



Their product DFA gives their intersection: ACACAAC and CCCACCC.



We observe that computing the product DFA is alphabet-dependent, due to branching (transition function) on the same letter in the states of the two input DFAs.